COST REDUCTIONS FROM MULTI-MISSION SEQUENCING SOFTWARE

Laura Needels
Jet Propulsion Laboratory
4800 Oak Grove Drive
M/S 301-250D
Pasadena, CA 91109-8099
Laura.Needels@jpl.nasa.gov

Abstract

Sequencing software for deep space missions has historically been one of the most critical parts of the ground software used to communicate with and control the spacecraft. At JPL, the sequencing software is responsible for planning and creation of science and engineering activities, checking command syntax, checking mission and flight rules, and translating the commands into packets which can be uplinked to the spacecraft. Significant effort has been spent by earlier missions to ensure the integrity of this software since errors in this area could cause the spacecraft to enter fault protection or cause the loss of the spacecraft.

Over the last several years, in an effort to reduce the costs associated with sequencing software, the Advanced Multi-Mission Operations System (AMMOS), part of the Interplanetary Network Directorate (IND) at JPL has taken two steps to reduce the cost of the sequencing software. The first step was to develop a multi-mission form of sequencing software. The second step was to develop a set of automation scripts which permit of automation scripts which permit 24x7 commanding with little human intervention.

The development of the multi-mission software began about 10 years ago. The architecture applied to the multi-mission software is to develop it as two separate components. One component, the multi-mission "core" software provides, in a generic sense, the capability to perform the functions needed in the sequencing software: planning and scheduling events, checking flight rules, and packetizing commands. The "core" software is then "adapted" to a project specific mission. The "adaptation" part of the software task involves providing the models for activities needed for planning and scheduling, converting the command list for the project into models that can be used for sequence checking, coding project and mission flight rules and the modeling needed to support them, and developing project blocks for repetitive activities. The AMMOS software has been or is currently being used by Mars Global Surveyor, Cassini, Deep Space 1, Ulysses, TOPEX, Mars Climate Orbiter. Mars Polar Lander, NEAR, Stardust, Mars Odyssey, JASON, Genesis, CONTOUR, Space InfraRed Telescope Facility, Mars Exploration Rover, Deep Impact, and Mars Reconnaissance Orbiter.

By partitioning the sequencing software into a "core" component and an "adaptation" component, JPL has been able to reduce costs. The core component of the software is verified once, by a central group, rather than by multiple projects. Projects can then focus on verification of just the adaptation part of the software. An additional cost saving with this methodology has been that it has allowed JPL to staff a multi-mission adaptation team where personnel can move from one project to another with minimal start-up and training time.

The automation scripts began development in the mid 1990's. Faced with the Faster, Better, Cheaper mantra, operations costs were forced to come down. Development of an automatic process for generating, verifying, and uploading commands had several benefits. One is that remote users (e.g. scientists) could stay remote. A second is that many of the steps needed to verify and uplink a sequence could be run much more quickly than if people were running them. The third is that much of the operations staff was reduced because commanding could occur 24x7 without staff on shift.

This paper describes the architecture of the JPL multi-mission sequencing system and the sequence automation process, discusses the cost savings associated with both of these changes.

1. Introduction

At JPL, the system used to develop the sequences that contain the commands sent to the JPL interplanetary spacecraft consists of several different stages, sequence planning, sequence verification, and sequence translation into spacecraft understandable binary. The software used to support these activities is known as SEQ.

SEQ consists of about a dozen different program sets. It is not necessary for a project to use all of the SEQ programs. Interface specifications for each program are well defined, and if a project chooses to use a separate program for one part of the system, it can easily be accomplished. Most projects do have their own components in some part of the uplink process.

Two of the SEQ programs are used by scientists to plan and verify science observations. The first science-planning program is called Science Opportunity Analyzer (SOA). SOA has a very user-friendly interface that allows scientists and other planners to exercise trade studies and preliminary design for science observations. Search engines that will locate times of interest such as fly-bys, bow-shocks, occultations, angles, periapsis and apparent diameter are available. Once these opportunities have been located, the user can design a specific observation (e.g. Continuous Scan, Roll Scan, Start Stop Mosaic, Stare). Criteria such as the primary target, secondary target, target offsets, duration, primary observer, and secondary observer are entered. Constraint checking for activity duration, distance, exclusion zones, and hardware limits is also completed by SOA. SOA is such a flexible tool that lander missions are also considering the use of SOA to identify communication times with orbiting assets. The second program is called Planetary Observation Instrument Targeting and Encounter Reconnaissance (SEQ_POINTER, or just POINTER). Once the preliminary science design has been completed, the observations are passed to POINTER, where detailed calculations and constraint checking are completed. There are usually two differences between POINTER and SOA. The first is that SOA works at the activity level, but POINTER works at the command level. The second is that POINTER usually has a more complex spacecraft and/or turn model associated with its calculations than SOA. It is possible to implement the more sophisticated models in SOA, however this would slow down calculations. Since SOA is currently being used for preliminary science design activities, the more complicated models are not necessary at that stage of the planning.

The SEQ program used to monitor engineering resources at the activity level is called Activity Plan Generator (APGEN). The engineering planning is done to ensure that resources such as power, data storage, or reaction wheel revolutions per minute are not exceeded. ApGen allows the user, more often a mission planner or a spacecraft engineer, rather than a scientist, to plan activities while monitoring resource constraints. Desired activities might be Deep Space Network (DSN) contacts, science activities, and general engineering activities. The resources monitored might be solid-state recorder space, propellant, and battery state of charge. The model fidelity of both the activities and the resources increases during mission development. For example, in the beginning of mission development, science activities might be assumed to use an assumed level of power and output an assumed level of data on the solid-state recorder. As the mission design becomes more exact, different science activities may be modeled with different power and data volume needs. Information about science activities might even come from SOA or POINTER since communications exist between these programs. ApGen also contains a scheduler, which will allow a user to define activities, some of which are fixed in time (e.g. DSN contacts), and additional desired activities. The scheduler will place activities on the timeline so that resources are not violated. The scheduler will also show which of the desired activities could not be scheduled. The user can then reprioritize activities (only two Type A activities and three Type B activities) or add additional activities that would alleviate resource constraints (add an additional DSN contact so that space on the data recorder becomes available). One project has integrated a more sophisticated planner into ApGen. Work has also been proposed to modify the structure of ApGen to make it much easier to integrate whatever planner is desired.

SEQ offers a tool that will assist the user in developing sequences. Sequence Generator (SeqGen) is a tool that has three functions. One function is to help the user generate a sequence. A GUI is provided that allows the user to select a command and input or select parameters for a command. Regardless of whether the SeqGen GUI is used to generate sequences, SeqGen does provide command syntax checking, such as the number of input parameters and the parameter types. The second function is to integrate sequences. These sequences might be any combination of background, science, or engineering sequences. The third function is to verify sequences. The most common use of SeqGen is for sequence integration and verification, since sequences are often developed by other programs (such as ApGen, POINTER, or SOA) or tools that scientists or spacecraft engineers have created to meet specific needs. When sequences are integrated, it is imperative to check that mission rules and flight rules are not violated by the combined sequences. Projects try to implement as much rule checking as possible in SeqGen because it is much more efficient and less tedious than using manual procedures to check the rules. Historically, the mission and flight rules checked by SeqGen were more timing related, such as “The cat bed heaters must be turned on 5 minutes before the thrusters are fired,” or “Once the spacecraft has been launched, the mission phase may never be set to pre-launch.” SeqGen has also been used for checking commands against spacecraft states, such as “The command may be used only while the spacecraft is nadir pointed,” or “The command may not be issued while the spacecraft is in eclipse.” However, more recently, SeqGen has also been used to check for resource constraints using models at the command level. An example of this is using an externally provided power model to monitor the battery state of

charge. Another project has integrated a sophisticated slew model to provide time estimates for slew activities. Output from SeqGen consists of a predicted events file, a human readable spacecraft sequence file that contains the commands that will be sent to the spacecraft, a file containing spacecraft states, and a log file.

Other SEQ programs are used to translate the human readable spacecraft sequence file into the binary which is sent to the spacecraft, a program that is used to assist project adapters build model files needed by SeqGen, a program which parses predicted events data, allowing the user to generate strips of data the user deems interesting, and spacecraft sequence of events files.

2. Multi-mission Architecture

When the Faster, Better, Cheaper mantra became a requirement, the strategy of software reuse became popular. Since the time and money involved in generating sequence software is significant, a framework was developed that allowed for sequence software reuse was developed. SEQ was partitioned into two parts, Core and Adaptation. SEQ Core software has the ability to perform a specific function. SEQ Adaptation provides project specific components that rely on core functions.

An analogy for this sort of division can be seen in tool used for word processing. In SEQ terminology, the word processing software is piece of Core Software. The word processing software allows the user the ability to pick various font types (Times, Ariel, Courier, and Symbol), font representations (bold, italic, and underlined), and paragraph alignments (left justified, right justified, and center justified). And it is the user (the Adapter) who writes the text, chooses the styles and decides the purpose of the document (a letter or a memo).

An example of core functionality is SeqGen's ability to check flight rules. Adapters model spacecraft states. When a spacecraft command causes a rule to be violated, SeqGen issues an alert. Similarly, adapters model spacecraft resources, and ApGen issues an alert when an activity causes a spacecraft resource to go outside established limits.

3. Multi-mission Architecture Cost Savings

The multi-mission architecture offers advantages and cost savings through several different ways. The single biggest cost savings comes from the fact that capability is both implemented and verified only once. Another cost savings comes from team familiarity with the software. A third advantage is that teams (flight software development and Assembly, Test and Launch Operations teams) are ready to hit the ground running because a large portion, if not all, of the capability needed is already available in the software. Another advantage is that the software is inherently more reliable because characteristics are well known (and don't appear as surprises).

The multi-mission architecture provides cost savings to projects because many of the needed capabilities are already developed. One recent project needed only very small (less than two work months) additional capability added to the ground system uplink software. The cost savings from the development of the core software are not always immediately apparent to projects. One recent project thought the costs for the adaptation of the multi-mission software were high until it estimated from historical databases and models how much it would take to start from scratch and implement project specific software. This project even needed a significant number of changes implemented in the core software to support some of the mission operations desires. Given the number of changes needed by this project, this would have been a case that tested the limits of the value of already having software in place with part of the capability.

A Return On Investment was done on the overall savings of the entire AMMOS system. The first part of the cost savings in is that development does not have to be done. Mission specific adaptation costs of multi-mission tools were calculated to be ~15% of mission specific development costs. Very simply, project cost savings are equal to the amount of money it would have taken to develop their ground system not using AMMOS multi-mission software, minus the project adaptation costs, minus the cost of any changes that need to be added to the core multi-mission software.

Cost_Savings = Cost_No_Reuse – (Cost_Adaptation + Project_Specific_Changes)

The models used estimate this savings consist of low-cost, medium-cost, and flagship missions. The second part of the cost savings is in software verification. Software verification has to be completed only once for a majority of the functionality. Projects still must verify the project specific adaptation (command parameters, rule modeling, etc.), but this would have to be done regardless of whether AMMOS multi-mission software is used. Core software verification in SEQ at both the program level and the subsystem level currently takes about 6 weeks with some of the program level and subsystem level testing done in parallel. On current missions, a single project's adaptation specific testing is completed in less than a week. During the Y2K frenzy, 9 projects were using multi-mission software, and rather than 9 separate verification efforts, only one had to be done. The overall Return on Investment study, which includes costs to maintain the software, new capability development (that is considered to be a multi-

mission capability), continuous improvement programs, systems engineering, development platform costs (development software licenses, hardware replacement, software development and test facilities), and configuration management found that $1 invested in AMMOS multi-mission software saves NASA Code S (projects) $3. [1]

A second cost savings comes from team familiarity with software. This comes in two flavors. The first flavor is that personnel from one project can easily transition to another project because the software used is identical. While project specific implementations (rules, models, etc.) may vary, the framework and syntax of the software is known. This also means that it is easy to move staff from one project to another during critical events (additional support to meet a launch or other mission critical deadline, during adaptation intensive times caused by planned or unplanned events, etc.) The second flavor is that personnel within the project use the same software in all phases of the mission, so transition costs from Development to Operations phases do not exist.

An additional advantage in the use of multi-mission software is that Flight Software (FSW) and Assembly, Test, and Launch Operation (ATLO) teams can hit the ground running. A major portion of the needed capability exists. A ground system implementation with a simple NOOP command can be developed in less than a week. Similarly, once a few commands are known, simple ground system implementations can rapidly developed to support rapid prototype testing of FSW and in ATLO. FSW and ATLO teams have even been trained to convert Command Dictionaries into use by SeqGen when rapid development is done.

The last advantage of multi-mission software discussed in this paper is that the ground system software is more reliable and offers more functionality than could be obtained with single use ground system software. Continuous maintenance, which includes bug fixes and small enhancements, results in highly reliable tools. Projects also often get more functionality using multi-mission software than could be obtained with mission specific (single use) ground software. Since a large portion of capability is on the shelf already, projects can choose to add extra features that could not have been implemented because of time or monetary constraints. [1].

4. Automated Sequence Processor

The Automated Sequence Processor (ASP) is a collection of scripts that were developed to automate the sequence generation process during the aerobraking phase of the Mars Global Surveyor mission. Aerobraking is used to gradually lower the spacecraft into a circular orbit. This is done with “drag pass” sequences. The spacecraft is lowered into the atmosphere. However, because the Martian atmosphere is variable (thermal changes cause it to bloom), the orbit of the spacecraft must be continually monitored to make sure the spacecraft does not go too far in or out of the atmosphere. The aerobraking maneuvers to raise or lower the orbit can be commanded every orbit. Science observations to monitor the orbit may also be performed every orbit. As the spacecraft orbit gets lower, orbit times shorten, and the time for each orbit can be as low as two hours. That means that the new command load must be prepared in a very short time. Before the ASP automation process was begun, it took two people approximately two hours to generate the necessary command products for a single command request file containing multiple commands. In order to support the aerobraking process, something had to be done to speed up the sequence generation process.

Although the automation process was begun to support the aerobraking requirements of Mars Global Surveyor in the mid 1990s, many other benefits of the effort have become apparent. One of the most obvious is that operations team costs have gone down because fewer people are needed to support a mission. Through additional development on the ASP, capability has been added so that remote users such as spacecraft contractors or science teams can work from their home institutions. Another advantage is that automatic verification of part of the sequence generation process is more reliable. Finally, some types of commanding can be done completely automatically, meaning the operations staff was reduced because commanding could occur 24x7 without staff on shift.

Prior to the implementation of the ASP, a requestor (a principal investigator or spacecraft team member) would submit a command file to the Sequence Team for processing. Two project sequence team operations members would manually use team procedures and software tools to generate a validated the necessary uplink products. This took an average of two hours. Currently, the requestor submits the command file to the ASP for processing. The ASP executes all necessary tasks to generate the uplink products for transmission. Processing though the ASP in all circumstances except one special set of cases of processing takes approximately five minutes. In the one special set of circumstances, very special commanding is being done, so the additional verification steps are taken to make sure that the command load will not harm the spacecraft. [2].

A trigger mechanism is used to kick off the ASP processing. The original trigger mechanism was an email containing the file release form that holds information about the command file that will be processed. Recently, this trigger mechanism was replaced with a more robust

mechanism that involves publishing the file release form information to a database that is periodically queried to see if a new file has arrived. Once the command processing has begun, scripts are use to mimic the manual generation process. This includes incrementally checking product generation. Command products are also stored on the project database, and moved out to Deep Space Network for eventual radiation. Many of the scripts used by the ASP are the scripts that would have been used during manual sequence generation. The ASP provides the glue to patch the steps together, which consists primarily of verification of steps completed as well as the criteria to determine which step should come next. Naturally verification to ensure a requestor is a legitimate user is also done.

Since the ASP allows users to submit command files from remote sites, and the ASP does user verification, remote users do not have to maintain a presence at Jet Propulsion Laboratory (JPL) to complete commanding activities. This saves a significant amount in travel expenses and greatly improves the morale of remote users.

The ASP has an added safety benefit because much of the checking is done via an automatic process that is less susceptible to error than manual checking. In 2002, approximately 9000 files were processed. Of those, there were no command errors caused by ASP processing. Eliminating some of the drudgery and difficulty of verifying command products helps teams generate better command products.

Finally, the automatic nature of the ASP allows commanding to be done day or night by science teams without sequence support staff on hand. Some types of spacecraft commanding can also be done 24x7 without sequence teams on hand.

5. Automated Sequence Processor Cost Savings

Implementation of the Automatic Sequence Processor has provided cost savings in several different ways. The most tangible ways have been a reduction in the size of Sequence teams. Other intangible ways include better integrity of command products and better freedom and morale for non-JPL teams that commanding responsibilities. The ASP has also allowed JPL to complete mission phases that could not be completed without automation.

A Return On Investment was recently completed for the Automatic Sequence Processor. The Return On Investment included only the tangible cost savings of the reduction in Sequence Team size. In 2002, since there were almost 9000 command files were processed through the ASP, this would have required approximately 36,000 work hours without the ASP. This is roughly $3.4 million dollars that is not spent on Sequence teams every year. The cost of the ASP was the initial expenditure, maintenance costs, and upgrade costs done in the last year. The initial and upgrade cost is approximately $250,000. Daily ASP maintenance is also performed by the Sequence team at a cost of approximately $40,000 per year. [2]

The Return On Investment calculation described above does not include the adding different types of processing capability to the ASP. Nor does it include the cost of adding new projects to the ASP. If a new project wants to use the ASP, and uses existing Sequence Team processes to generate command files, it takes $10,000 - $20,000 to give them ASP processing capability.

6. Conclusions

Two steps have been taken in the last several years to reduce costs in the Uplink portion of the Ground Data System used by JPL. The first step, converting to a multi-mission architecture for the Sequencing software, has offered not only reduced costs for subsequent projects, but also improved reliability for projects. The second step, implementing an Automated Sequence Processor, has offered significant cost savings to projects while also reducing processing errors.

The argument for using a multi-mission architecture becomes more compelling for each additional project that is expected to use the software. The implementation of an Automated Sequence Processor can offer substantial savings, even if it used by only one project.

References

[1] Klose, J. Charles. Advanced Multi-Mission Operations System (AMMOS), Return On Investment Study Results. May, 2002.

[2] Reichert, R., R. Brooks, L. Needels, R. Thomas. Automated Sequence Process (ASP), Return on Investment. August, 2002.

Acknowledgements

The research described in this publication was carried at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.